cs151S11proj2-Lauren Zion

Lauren Zion

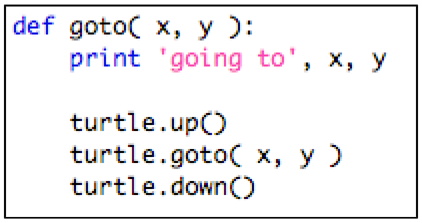
February 15, 2011

Overview:

Being the second project, this set of tasks built upon what we had learned in the first project. This time though, we were expected to create many shapes in one file that could be called and manipulated in another file. By defining these shapes using variables, their size, look, and position changed by altering the value of each variable. This way, very few shapes could be used together to create complex scenes. In our case, these scenes represented certain places or images of Colby.

My thought process:

First, in lab and on my own, I created a sort of dictionary of shapes that I could call back to. Before even creating these shapes, I had to create a "goto" function that allowed me to plug in certain x and y coordinates so that the shape would start being drawn at that point. The key to creating this function was to lift the pen up in order to avoid drawing lines all over the place while moving from point to point.



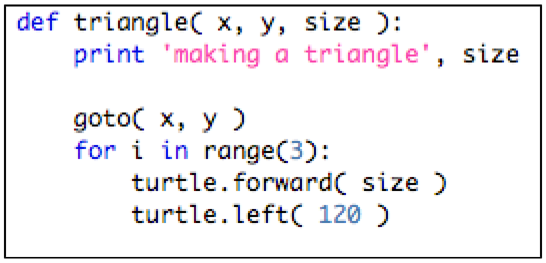
```
def goto( x, y ):
    print 'going to', x, y

    turtle.up()
    turtle.goto( x, y )
    turtle.down()
```

Fig. 1: This is a code snippet showing how I created the "goto" function using the variables x and y. (shapes.py)

***Earlier in the code, turtle was imported using "import turtle". It is important to import all modules at the beginning of the code so their functions can be referred to later. By using the command "import turtle", I had to remember to write "turtle." before any turtle commands I wanted to execute.

Next, I began to create definitions to make basic shapes. These included a square (block), an equilateral triangle, a circle, a semicircle, and a trapezoid. For the square and equilateral triangle, a for loop was used to limit the amount of code and also decrease the chances of making a mistake while writing the code. For the block, the sequence of forward(width), left(90), forward (height), and left(90) was repeated twice. The image below shows how a for loop was used to make the equilateral triangle. Because with an equilateral triangle you are doing the same functions over and over, the for loop is a nice way to decrease the number of lines of code and avoid mistakes.



```
def triangle( x, y, size ):
    print 'making a triangle', size

    goto( x, y )
    for i in range(3):
        turtle.forward( size )
        turtle.left( 120 )
```

Fig 2. This is a code snippet showing how a for loop was used to create an equilateral triangle. (shapes.py)

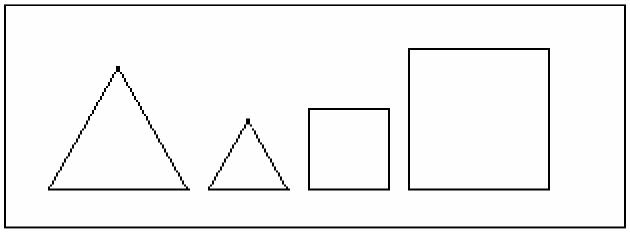


Fig 3. This is a line of some of the basic shapes I made. As you can see, the size of both the square and triangle can be easily manipulated.

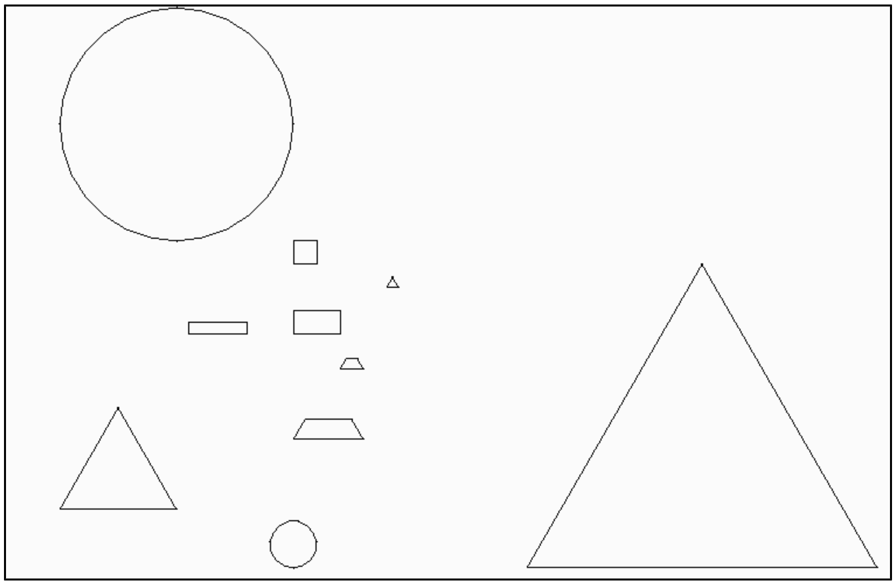
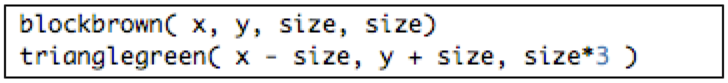


Fig 4. This image is a picture of more shapes being manipulated. This time, the y variable is also changed so that the shapes ended up all over the screen instead of in a line.

Once the basic shapes I needed were completed, I used them to make a few objects I knew I would be using a lot in my scenes. In order to make these objects, multiple shapes were combined together in a precise order. The easiest way to do this was to first draw the object I wanted to make on a piece of graph paper, and then try to write a code for it. In order to write the code, I had to treat the first point x and y as the main variable and add numbers or variables to it in order to place shapes in the correct places. This may sound confusing, but it is actually pretty simple. Say I wanted to make a shape that put two 1X1 boxes on top of each other. I would first make one box at the (x, y) point and use the width and height as the other variables. Next, in order to make the second box end up on top of the first and still only use the four variables, the x variable would remain the same for this box as the first, but the y variable would change so that the box started right above the first. This would be done by entering " y + height " in the y variable slot of the box. Maybe a code snippet with help explain this better...



```
blockbrown( x, y, size, size)
trianglegreen( x - size, y + size, size*3 )
```

Fig 5. This is a code snippet showing how the same four variables can be used to combine shapes in order to make one object. This example is making a green equilateral triangle directly above a brown block one-third of its size.

Using this technique, I made a bench and a tree. The bench was made all out of boxes, while the tree incorporated a block and three triangles in its design.

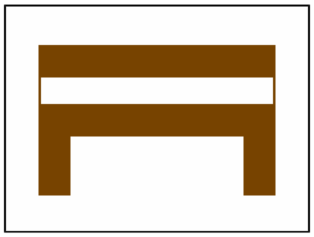


Fig 6. This is a picture of the bench made up of 5 blocks. (shapes.py)

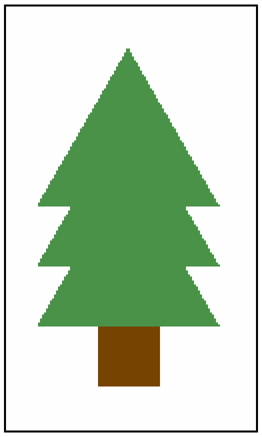


Fig 7. This is a picture of the (evergreen) tree made by combining one block and three equilateral triangles. (shapes.py)

With these shapes, I either wanted to make one in a certain location or a whole bunch of them over a certain spot in my scene. To do the latter, I had to import the random package using "import random". This module allows us to create a lot of the same shape in varying sizes and locations without writing a unique code for each one. To create the "forest" of trees seen below, I had to combine the use of a the random package with a for loop. I randomized all of the variables of the tree using the command "random.randint( )", and then picking the range of values within the parentheses. In order to make many many trees, I created a for loop that repeated the creation of a random sized and placed tree over and over again.

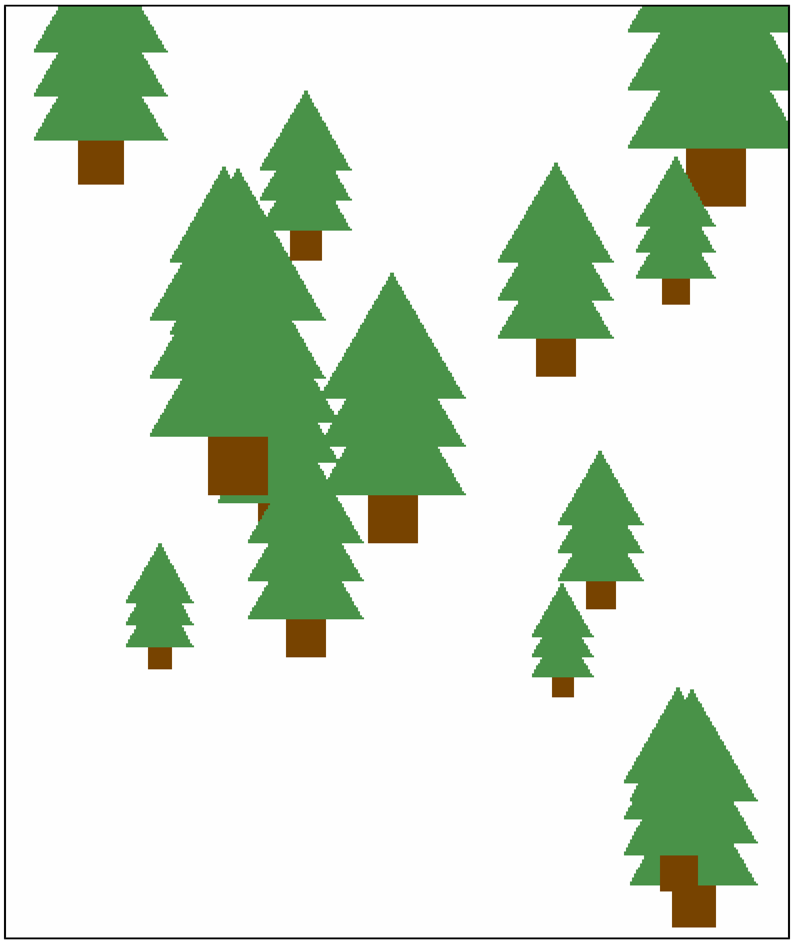
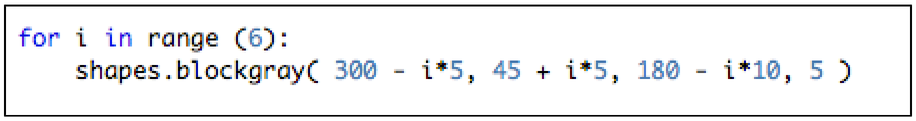


Fig 8. This is a forest of trees made by combining a for loop with the random package.

Now that I had my basic shapes and objects ready to go, I began to make my first scene: the Olin building at Colby. This was harder than I thought. I first created the building using just the basic black and white shapes. It was hard to line up every window and every stair. Looking back on it with the knowledge I have gained now that I finished this project, there are a lot of things I could have done to simplify my code. For example, I made the stairs block by block, stair by stair, while instead, I could have created a for loop which took into account the space between the start of the first shape and the start of the second shape. In my second image, I took my own advice and did use a for loop to create the bleachers (this code can be seen in Fig 9.). After getting the black and white sketch the way I wanted it, I next added color. Once again, looking back, my life could have been much easier if I had used variables to change the colors of each shape. Instead, I created a new shape for each color I needed it to be. This can be seen in my code because I have block blue, block gray, block red, and so on. Next time, I would certainly rectify this in order to shorten my code and simplify it.



```
for i in range (6):
    shapes.blockgray( 300 - i*5, 45 + i*5, 180 - i*10, 5 )
```

Fig 9. This is a code snippet that I created to make the bleachers in scene 2. The craziest part of this code is that it only took two lines of code to make the entire set of bleachers. The same code could have and should have been used to make the stairs leading up to the doorway in scene 1. (field.py)

After getting all the shapes in place, I must say that a hard part of the project was getting the colors to be exactly right using the RGB distribution. I also soon realized that the order of the shapes had a big impact on what showed up on top of what as I added colors to all my shapes. In other words, I had to lay the red brick of the building before laying the windows so that they would not be covered by the brick. Another thing that was hard to do was getting the placement of each shape to be exactly right to create symmetry and accuracy. I did use some guess and check, but the more code I wrote the better I got at estimating where a certain window or tree needed to be. Below is a picture of the real Olin building, and underneath that is my homemade version.

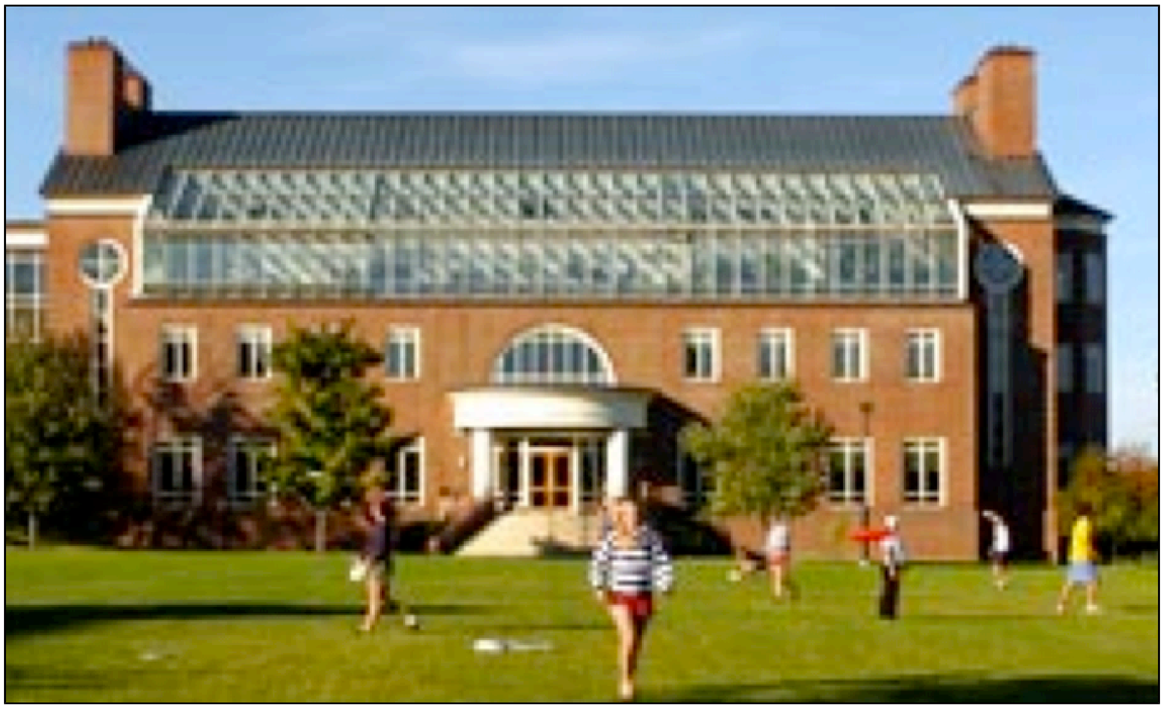


Fig 10. This is picture of the real Olin building at Colby. (google.com)

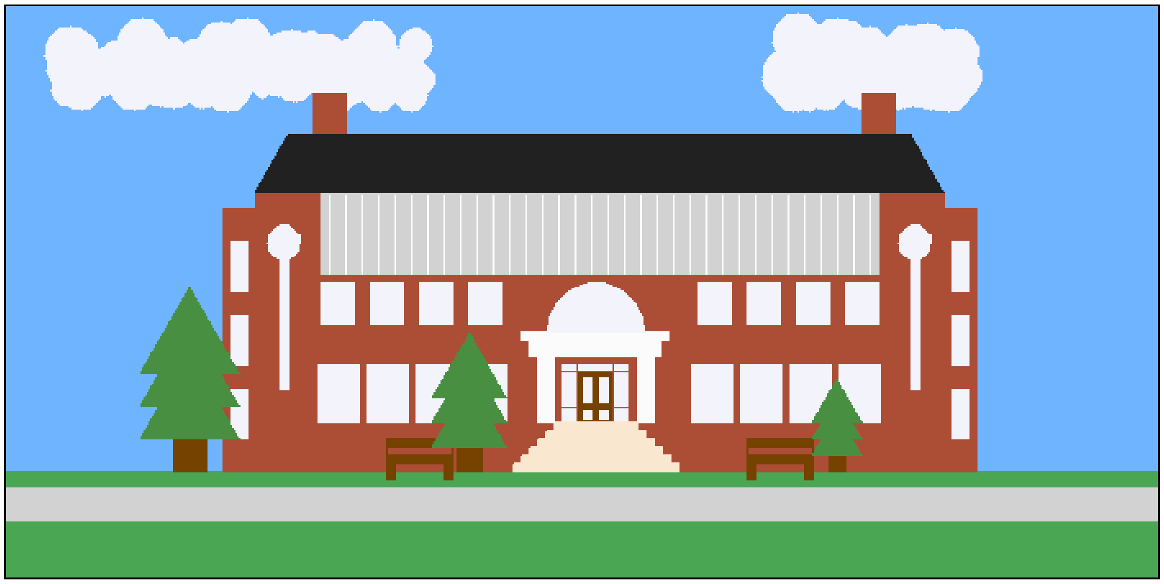


Fig 11. This is a screen capture of my version of Olin made using my shapes and objects found in the shapes.py file. (main.py)

*** Another important thing to remember that in order to use the shapes created in the shapes.py file, I had to import the file using "import shapes" at the beginning of my code. And then, whenever using a function from this file, I had to write "shapes." before writing the function names and entering the variable values.

Next, I used the same shapes and objects in order to create an entirely different picture. I decided to make the Colby field hockey field. The challenges I faced with this picture were very much the same as with scene 1, but this time I had a hard time getting the lines of the field to show up and be a certain color. I eventually found that by placing the creation of the lines after the function to make an object of the same color, the lines stayed white. I am sure there is an easier way to do this, and I am ?pretty sure the command is turtle.pencolor(r, g, b). I will be sure to use this function in my next code.

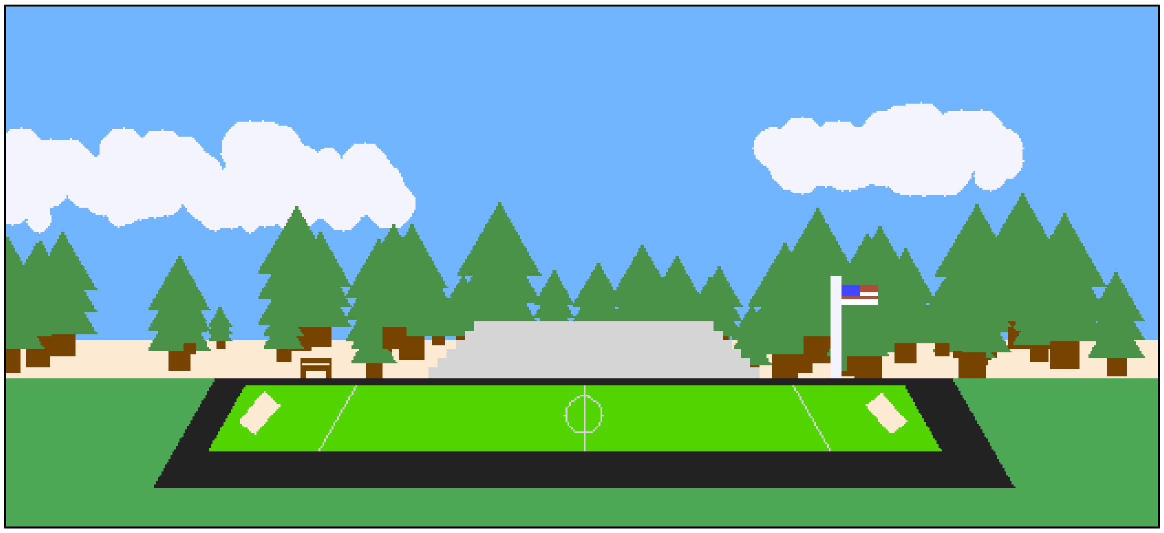
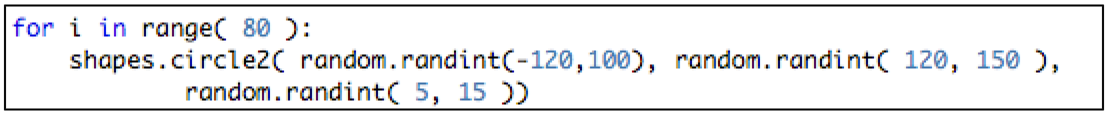


Fig 12. This is my version of the field hockey field here at Colby. (field.py)

As you can see in the picture above, there exists a forest of trees. Like the forest of trees in Fig 8., I used the random function combined with a for loop to created the sea of trees that you see. There are good things and bad things about the random function. I found that it made my life a lot easier to use this function to create the forest, but the downside was that sometimes the randomization would create something not too aesthetically pleasing. For example, the trees would all clump in one spot or only the big trees could be seen because they covered all the smaller trees. Well, in the end it added a lot of depth to this picture and I think the forest came out great. A for loop was also used to create the bleachers and can be seen in Fig 9..

You may be wondering how I made the clouds, and it is much simpler than you think. All I did was use the same techniques I used to make the forest but apply them to circles. I randomized the size and located of the circles within a certain area and than ran that code on a for loop to create a fluffy cloud. I ran this code 80 times to create a full and voluminous cloud. The code snippet can be seen below.



```python
for i in range( 80 ):
    shapes.circle2( random.randint(-120,100), random.randint( 120, 150 ),
            random.randint( 5, 15 ))
```

Fig 13. This is a code snippet showing how I made the cloud seen in the left-hand corner in Fig 12. (field.py)

What I learned!:

Overall, this entire project was a learning experience. I learned a great deal as I went along, if I redid the entire project having now finished it, I am sure that I could write a much more concise code while still making as complicated if not more complicated shapes. Some of the most interesting things I learned were how to properly use a for loop, how to randomize something, and how to create definitions of shapes that I could call to at any time.

Extensions:

The extensions can be found throughout the section above title "My thought process". These include:

- the use of color as a turtle property
- the use of multiple for loops
- importing the random package
- the use of randomness multiple times
- an additional forest scene combining a for loop and randomness
- using the fill capabilities with color as turtle properties

All files can be found in the academic server, under lezion within the folder titled Project 2.